ELSEVIER

Journal of Systems Architecture 46 (2000) 627–639

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Techniques for mapping tasks to machines in heterogeneous computing systems ☆

Howard Jay Siegel *, Shoukat Ali

School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA

Abstract

Heterogeneous computing (HC) is the coordinated use of different types of machines, net-works, and interfaces to maximize their combined performance and/or cost-effectiveness. HC systems are becoming a plausible technique for efficiently solving computationally intensive problems. The applicability and strength of HC systems are derived from their ability to match computing needs to appropriate resources. In an HC system, tasks need to be matched to machines, and the execution of the tasks must be scheduled. The goal of this invited keynote paper is to: (1) introduce the reader to some of the different distributed and parallel types of HC environments; and (2) examine some research issues for HC systems consisting of a network of different machines. The latter purpose is pursued by considering: (1) the quantification of heterogeneity; (2) the characterization of techniques for mapping (matching and scheduling) tasks on such systems; (3) an example HC resource management system; and (4) static and dynamic heuristics for mapping tasks to machines in such HC systems.

Keywords: Distributed computing; Heterogeneous computing; Metacomputing; Resource management system; Scheduling

1. Introduction

In general, heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to maximize their combined performance and/or cost-effectiveness [23,9]. HC systems are becoming a plausible technique for efficiently solving computationally intensive problems [12]. The applicability and strength of HC systems are derived from their ability to match computing needs to appropriate resources. As machine architectures become more advanced to obtain higher peak performance, only a fraction of this performance may be achieved on many real tasks because a typical task may have various subtasks with different architectural requirements. When such a task is executed on a given machine, the machine may spend much of its time executing subtasks for which it is unsuited [15].

One way to exploit an HC environment is to decompose a task into subtasks, where each subtask is computationally well suited to a single machine architecture, but different subtasks may have different computational needs (e.g., [29]). These subtasks may share stored or generated data, creating the potential for inter-machine dependencies and data transfer overhead. Once the

☆ This research was supported by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-98-M-0217 and N62271-98-M-0448, and under GSA subcontract number GS09K99BH0250. Some of the equipment used in this research was donated by Intel and Microsoft.

* Corresponding author.

E-mail addresses: hj@ecn.purdue.edu (H.J. Siegel), alis@ecn.purdue.edu (S. Ali).

1383-7621/00/$ - see front matter
PII: S1383-7621(99)00033-8

subtasks are obtained, each subtask is assigned to a machine (*matching*). Then the subtasks and any inter-machine data transfers are ordered (*scheduling*) so as to optimize some objective function. The overall problem of matching and scheduling is referred to as *mapping*. The objective function can be the overall completion time of the task or a more complex function of multiple requirements.

In some cases, a collection of independent tasks must be mapped, instead of a set of interdependent subtasks. Such an independent set of tasks is called a *meta-task* [14]. An example of meta-task mapping is the mapping of an arbitrary set of independent tasks from different users waiting to execute on a heterogeneous suite of machines. Each task in the meta-task may have associated requirements, such as a deadline and a priority.

One broad objective of the HC community is to design a management system for HC resources (machines, networks, data repositories, etc.) [23]. One important issue within this arena is the design of a mapping system that makes good decisions. Such a system, which may be called a *scheduling advisor*, has to be provided with an objective function that it tries to optimize. Current research involves formulating an optimization criterion that will be a function of a set of quality of service (QoS) attributes that are likely to be requested by the tasks expected in a given HC environment [21]. This optimization criterion also will serve as a measure of the performance of the various scheduling approaches that might be available to the community, and also for that of the resource management approaches in general.

The scheduling advisor might have to choose between static and dynamic approaches to the mapping of tasks. Static approaches are likely to suffice if the tasks to be mapped are known beforehand, and if the predictions about the HC resources are likely to be accurate. Dynamic approaches to mapping are likely to be more helpful if the HC system status can change randomly, and if the tasks that are supposed to be mapped cannot be determined beforehand. The general problem of developing an optimal matching of tasks to hosts is NP-hard [10]. The goal of this invited keynote paper is to: (1) introduce the reader to some of the different distributed and parallel types of HC environments; and (2) examine some research issues for HC systems consisting of a network of different machines. The latter purpose is pursued by considering: (1) the quantification of heterogeneity; (2) the characterization of techniques for mapping tasks on such systems; (3) an example HC resource management system; and (4) static and dynamic heuristics for mapping tasks to machines in such HC systems.

Section 2 briefly describes some broad classes of HC systems. In Section 3, 'mixed-machine' HC systems are further classified and elaborated. Section 4 characterizes heuristics for mapping independent tasks onto a class of HC systems. An example system for managing resources in HC systems is discussed in Section 5. Sections 6 and 7 define and compare some dynamic and static mapping heuristics, respectively. Section 8 concludes the paper.

2. Parallel and distributed heterogeneous computing systems

There is a great variety of types of parallel and distributed heterogeneous computing systems. In this section, three broad classes are briefly described: mixed mode, multi-mode, and mixed-machine. The rest of the paper focuses on mixed-machine systems.

A mixed-mode HC system refers to a single parallel processing system, whose processors are capable of executing in either the synchronous SIMD or the asynchronous MIMD mode of parallelism, and can switch between the modes at the instruction level with negligible overhead [27]. Thus, a mixed-mode machine is *temporally* heterogeneous, in that it can operate in different modes at different times. This permits different modes of parallelism to be used to execute various portions of a program. The goal of mixed-mode HC systems is to provide in a single machine the best attributes of both the SIMD and the MIMD models. PASM, TRAC, OPSILA, Triton, and EXECUBE are examples of mixed-mode HC systems that have been prototyped [27].

There are various trade-offs between the SIMD and MIMD modes of parallelism, and mixed-

mode machines can exploit these by matching each portion of a given program with the mode that results in the best overall program performance. Studies have shown that a mixed-mode machine may outperform a single-mode machine with the same number of processors for a given program (e.g., [11]).

Multi-mode HC is similar to mixed-mode HC in the sense that multiple modes of computation are provided within one machine. However, it is different because all modes of computation can be used simultaneously. An example multi-mode architecture is the image understanding architecture (IUA) [30]. In IUA, heterogeneity is incorporated by having multiple processing layers, where each layer provides a different form and mode of computation. Two levels of MIMD and one level of SIMD processors are included in this system.

Thus, mixed-mode and multi-mode systems represent one extreme of HC, where the heterogeneity resides in a single machine. For more about such systems, see [9].

In mixed-machine HC systems, a heterogeneous suite of machines is interconnected by high-speed links to function as a metacomputer [20] or a grid [12]. (The grid refers to a large-scale pooling of resources to provide dependable and inexpensive access to high-end computational capabilities [12].) A mixed-machine HC system coordinates the execution of various components of a task or a metatask on different machines within the system to exploit the different architectural capabilities available, and achieve increased system performance [23].

3. Degrees and classes of mixed-machine heterogeneity

In a mixed-machine HC system, each task can have a different execution time on each machine. A heuristic is employed to map tasks onto the machines in an HC system. The assumption that estimates of expected task execution times on each machine in the HC suite are known is commonly made when studying mapping heuristics for HC systems (e.g., [16]). (Approaches for doing this estimation based on task profiling and analytical benchmarking are discussed in [23].) These estimates can be supplied before a task is submitted for execution, or at the time it is submitted. The mapper contains an expected time to compute (ETC) matrix that contains the expected execution times of a task on all machines, for all the tasks that are expected to arrive for service. In an ETC matrix, the elements along a row indicate the execution times of a given task on different machines, and those along a column give the execution times of different tasks on a given machine. The average variation along the rows is referred to as the machine heterogeneity; this is the degree to which the machine execution times vary for a given task, averaged over all the tasks [2]. Similarly, the average variation along the columns is referred to as the task heterogeneity; this is the degree to which the task execution times vary for a given machine, averaged over all the machines in the system [2].

Based on the above idea, four categories were proposed for the ETC matrix in [2]: (a) high task heterogeneity and high machine heterogeneity (HiHi), (b) high task heterogeneity and low machine heterogeneity (HiLo), (c) low task heterogeneity and high machine heterogeneity (LoHi), and (d) low task heterogeneity and low machine heterogeneity (LoLo). The ETC matrix can be further classified into two classes, consistent and inconsistent [2], which are orthogonal to the previous classifications. For a consistent ETC matrix, if machine m_x has a lower execution time than machine m_y for task t_k, then the same is true for any task t_i. The consistent case represents the situation where there is a definite ordering among the compute power of the machines in the suite. In inconsistent ETC matrices, the relationship among the execution times for different tasks on different machines is random. The inconsistent case represents a mix of task computational requirements and machine capabilities such that no ordering as that in the consistent case is possible. A combination of these two cases, which may be more realistic in many environments, is the semi-consistent ETC matrix, which is an inconsistent matrix with a consistent submatrix. As an example, in a given semi-consistent ETC matrix, 50% of the tasks and 25% of the machines may define a consistent sub-matrix. These degrees and classes of

mixed-machine heterogeneity can be used to characterize HC environments.

4. Characterizing mapping heuristics

The mapping of tasks and meta-tasks, and the scheduling of communications in HC environments, are active, growing areas of research. A taxonomy of mapping heuristics is useful to researchers as it allows meaningful comparison among different mapping heuristics used in different HC environments for different applications. The Purdue HC Taxonomy [6] is a three part classification that attempts to classify mapping heuristics according to the features of the applications being mapped (i.e., application model), characteristics of the hardware that the mapper is targeting (i.e., HC hardware platform model), and mapping strategies (i.e., mapper model).

The Purdue HC Taxonomy classifies the application being mapped on the basis of latter's size, divisibility into (possibly dependent) subtasks, communication patterns, quality of service requirements, etc. The taxonomy distinguishes among hardware platforms on the basis of communication time estimates between different machines, possibility of concurrency in sends and receives, machine architecture and heterogeneity, interconnection network, etc. Similarly, mapping heuristics are classified on the basis of their ability to adapt to changes in HC system, support various application models, consider subtask data dependencies and communication times, fault tolerance, etc.

A researcher also can use the taxonomy to find mapping heuristics that use similar target platform and application models. The mapping heuristics found for similar models can then possibly be adapted or developed further to better solve the mapping problem under consideration.

5. MSHN: An example resource management system

5.1. Overview

A resource management system (RMS) views the set of heterogeneous machines that it manages as a single virtual machine, and attempts to give the user a location-transparent view of the virtual machine [24]. The RMS should be able to provide the users a higher level of overall performance than would be available from the users' local system alone.

The management system for heterogeneous networks (MSHN – pronounced 'mission') [18] is an RMS for use in HC environments. MSHN is a collaborative research effort that includes the Naval Postgraduate School, NOEMIX, Purdue University, and the University of Southern California. It builds on SmartNet, an operational scheduling framework and system for managing resources in an HC environment developed at NRaD [13].

The technical objective of the MSHN project is to design, prototype, and refine a distributed RMS that leverages the heterogeneity of resources and tasks to deliver the requested QoS. To this end, MSHN is investigating: (1) the accurate, task-transparent determination of the end-to-end status of resources; (2) the identification of different optimization criteria and how non-determinism and the granularity of application and platform models (as outlined by the Purdue HC Taxonomy [6]) affect the performance of various mapping heuristics that optimize those criteria; (3) the determination of how security should be incorporated within components as well as how to account for security as a QoS attribute; and (4) the identification of problems inherent in application and system characterization.

5.2. MSHN architecture

Fig. 1 shows the conceptual architecture of MSHN. As can be seen in the figure, every task running within MSHN makes use of the MSHN client library (CL) that intercepts the task's operating system calls. The scheduling advisor (SA) determines which set of resources a newly arrived task (or equivalently, a newly started process) should use. (Using the terminology from Section 1, the SA is a mapper.) The resource status server (RSS) is a quickly changing repository that maintains information concerning the current availability of resources. Information is stored in the RSS as a result of updates from both the CL

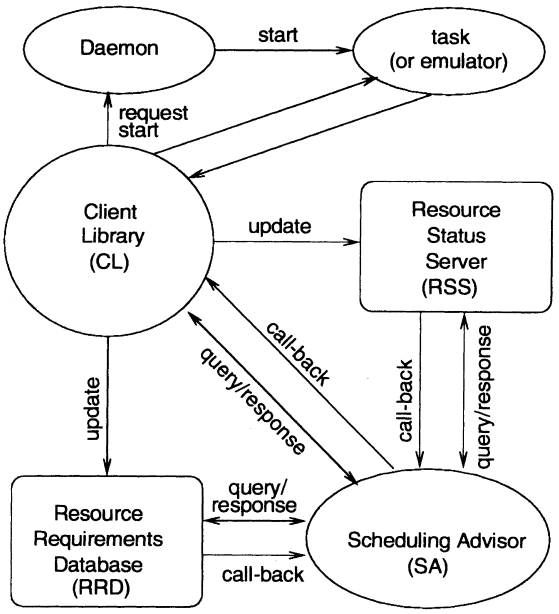


Fig. 1. High-level block diagram of the functional architecture of MSHN.

and the SA. The CL can update the RSS as to the currently perceived status of resources, which takes into account resource loads due to processes other than those managed by MSHN. The resource requirements database (RRD) is responsible for maintaining information about the resources that are required to execute a particular task. The RRD's current source of information about a task is the data collected by the CL from the previous runs of the task. The RRD has the ability to maintain very fine grain experiential information collected by the CL, and it is hoped that, in future, it can also be populated with information from smart compilers and directives from task writers.

When the CL intercepts a request to execute a new task, it invokes a scheduling request for that task on the SA (assuming that the task requests to be scheduled through the SA). The SA queries both the RRD and the RSS. It uses the received information, along with an appropriate search heuristic, to determine the resources that should host the new process. Then, the SA returns the decision to the CL, which, in turn, requests execution of that process through the appropriate MSHN Daemon. The MSHN Daemon invokes the application on its machine.

As a process executes, the CL updates both the RSS and the RRD with the current status of the resources and the requirements of the process. Meanwhile, the SA establishes call-backs with both the RRD and the RSS to notify the SA if either the status of the resources has significantly changed, or the actual resource requirements are substantially different from what was initially returned from the RRD. In either case, if it appears that the assigned resources can no longer deliver the required QoS, the application must be terminated or adapted (e.g., use an alternative implementation that may deliver less QoS, but requires less resources). Upon receipt of a call-back, the SA might require that several of the applications adapt so that more of them can receive their requested QoS.

5.3. Research issues for MSHN's scheduling advisor

The formulation of an optimization criterion for mapping tasks in complex HC environments is currently being researched in the HC community. Resource allocation involves attempting to solve heuristically an NP-complete optimization problem. MSHN is developing a criterion that maximizes a weighted sum of values that represents the benefits of delivering the required and desired QoS (including security, priorities, and preferences for versions), within the specified deadlines, if any. MSHN attempts to account for both preferences for various versions and priorities. That is, when it is impossible to deliver all of the most preferred information within the specified deadlines due to insufficient resources, MSHN's optimization criterion is designed to favor delivering the most preferred version to the highest priority applications. In MSHN's optimization criterion, deadlines can be simple or complex. That is, sometimes a user could use a piece of information only if it is received before a specific time. At other times, a user would like to associate a more general benefit function, which would tell how beneficial the information is to user, depending on when it is received. Further information about MSHN's optimization criterion can be found in [21].

The relative performance of search algorithms is another research issue. The MSHN team has obtained extensive results identifying the regions of heterogeneity where certain heuristics perform better than others for maximizing throughput by minimizing the time at which the last application, of a set of applications, should complete (e.g., [2,7,22]). Re-targeting of these heuristics to other optimization criteria is currently underway. Additionally, MSHN team members have performed extensive research into accounting for dependencies among subtasks (e.g., [3–5,29]). Sections 6 and 7 outline some of the MSHN research in the static and dynamic mapping of meta-tasks in HC environments.

6. Dynamic heuristics for mapping meta-tasks in HC systems

6.1. Overview

This section describes and compares eight dynamic heuristics that can be used in an RMS like MSHN for mapping meta-tasks [22]. In an HC system where the tasks to be executed are not known a priori, dynamic schemes are necessary to match tasks to machines, and to compute the execution order of the tasks assigned to each machine. A dynamic scheme is also needed in environments where some machines in the suite may go off-line and new machines may come on-line. These dynamic mapping heuristics are non-preemptive, and assume that the tasks have no deadlines or priorities associated with them.

The mapping heuristics can be grouped into two categories: on-line mode and batch-mode heuristics. In the on-line mode, a task is mapped onto a machine as soon as it arrives at the mapper. In the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at pre-scheduled times called mapping events. The definition of a meta-task can now be refined as the independent set of tasks that is considered for mapping at the mapping events. A meta-task can include newly arrived tasks (i.e., the ones arriving after the last mapping event) and ones that were mapped in earlier mapping events but did not begin execution. While on-line mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution.

The on-line heuristics consider, to varying degrees and in different ways, task affinity for different machines and machine ready times. The batch heuristics consider these factors, as well as aging of tasks waiting to execute. The ready time, r_k, of a machine m_k is defined as the earliest time that machine is going to be ready after completing the execution of the tasks that are currently assigned to it. It is assumed that each time a task t_i completes on a machine m_j a report is sent to the mapper. Because the heuristics presented here are dynamic, the expected machine ready times are based on a combination of actual task execution times and estimated expected task execution times. The experiments conducted in [22] to study dynamic mapping heuristics model this situation using simulated actual values for the execution times of the tasks that have already finished their execution. Also, all heuristics examined in [22] operate in a centralized fashion on a dedicated suite of machines; i.e., the mapper controls the execution of all jobs on all machines in the suite. It is also assumed that the mapping heuristic is being run on a separate machine. The next few subsections condense some discussions and results from [22].

6.2. Background terms and a performance measure for dynamic mapping heuristics

The expected execution time e_{ij} of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given m_j has no load when t_i is assigned. The expected completion time c_{ij} of task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let m be the total number of the machines in the HC suite. Let K be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of the task t_i be a_i, and let the begin time of t_i on m_j be b_{ij}. From the above definitions, $c_{ij} = b_{ij} + e_{ij}$. Let c_i be c_{ij}, where machine m_j is the one assigned by the mapping

heuristic to execute task t_2. The makespan for the complete schedule is then defined as $\max_{t_i \in K}(c_i)$. Makespan is a measure of the throughput of the HC system, and does not measure the quality of service imparted to an individual task. One other performance measure is given in [22].

6.3. On-line mode dynamic mapping heuristics

The minimum completion time (MCT) heuristic assigns each task to the machine that results in that task's earliest completion time. This causes some tasks to be assigned to machines that do not have the minimum execution time for them. The MCT is a variation of the fast greedy heuristic from SmartNet [13].

The minimum execution time (MET) heuristic (a variation of the user directed assignment in [13]) assigns each task to the machine that performs that task's computation in the least amount of execution time. This heuristic, in contrast to the MCT, does not consider machine ready times, and can cause a severe imbalance in load across the machines. The main advantage of this method is its simplicity.

The switching algorithm (SA) heuristic [22] is motivated by the following observation. The MET heuristic can potentially create load imbalance across machines by assigning many more tasks to some machines than to others, whereas the MCT heuristic tries to balance the load by assigning tasks for earliest completion time. The SA heuristic uses the MCT and MET heuristics in a cyclic fashion depending on the load distribution across the machines. The purpose is to have a heuristic with the desirable properties of both the MCT and the MET.

Let the maximum ready time over all machines in the suite be $r_{\max}$, and the minimum ready time be $r_{\min}$. Then, the load balance index across the machines is given by $\pi = r_{\min}/r_{\max}$. The parameter π can have any value in the interval [0, 1]. Two threshold values, π_l (low) and π_h (high), for the ratio π are chosen in [0, 1] such that $\pi_l < \pi_h$. Initially, the value of π is set to 0.0. The SA heuristic begins mapping tasks using the MCT heuristic until the value of load balance index increases to at least π_h. After that point in time, the SA heuristic begins using the MET heuristic to perform task mapping. This causes the load balance index to decrease. When it reaches π_l, the SA heuristic cycle continues.

The KPB (k-percent best) heuristic [22] considers only a subset of machines while mapping a task. The subset is formed by picking the (km/100) best machines based on the execution times for the task, where $100/m \leqslant k \leqslant 100$. The task is assigned to a machine that provides the earliest completion time in the subset. If $k = 100$, then the KPB heuristic is reduced to the MCT heuristic. If $k = 100/m$, then the KPB heuristic is reduced to the MET heuristic. A 'good' value of k maps a task to a machine only within a subset formed from machines computationally superior for that particular task.

The opportunistic load balancing (OLB) heuristic (used for comparisons in [13]) assigns a task to the machine that becomes ready next. It does not consider the execution time of the task when mapping it onto a machine. If multiple machines become ready at the same time, one machine is arbitrarily chosen.

6.4. Batch mode dynamic mapping heuristics

In the batch mode, meta-tasks are mapped after predefined intervals. For the ith mapping event, the meta-task M_i is mapped at time τ_i, where $i \geqslant 0$. The initial meta-task, M_0, consists of all the tasks that arrived prior to time τ_0, i.e., $M_0 = \{t_j \,|\, a_j < \tau_0\}$. The meta-task, M_k, for $k > 0$, consists of tasks that arrived after the last mapping event and the tasks that had been mapped, but did not start executing, i.e.,

$$M_k = \{t_j \,|\, \tau_{k-1} \leqslant a_j < \tau_k\} \cup \{t_j \,|\, a_j < \tau_{k-1},\ b_j > \tau_k\}.$$

The mapping events may be scheduled using one of the two strategies. The regular time interval strategy maps the meta-tasks at regular intervals of time. The fixed count strategy maps a meta-task M_i as soon as one of the following two mutually exclusive conditions are met: (a) an arriving task makes $|M_i|$ larger than or equal to a predetermined arbitrary number κ, or (b) all tasks have arrived, and a task completes while the number of tasks

which yet have to begin is larger than or equal to κ. In this strategy, the length of the mapping intervals will depend on the arrival rate and the completion rate. The possibility of machines being idle while waiting for the next mapping event will depend on the arrival rate, completion rate, m, and κ.

The Min-min heuristic shown in Fig. 2 is based on the ideas presented in [19], and implemented in SmartNet [13]. The Min-min heuristic calculates the minimum completion time for each task in the meta-task currently being considered. The task which has the minimum of these completion times is assigned the corresponding machine, and that machine's ready time is updated. This assigned task is removed from the meta-task and the procedure is repeated.

Min-min begins by scheduling the tasks that change the expected machine ready time status by the least amount that any assignment could. If tasks t_i and t_k are contending for a particular machine m_j, then Min-min assigns m_j to the task (say t_i) that will change the ready time of m_j less. This increases the probability that t_k will still have its earliest completion time on m_j, and shall be assigned to it. Because at $t = 0$, the machine which finishes a task earliest is also the one that executes it fastest, and from thereon the Min-min heuristic changes machine ready time status by the least amount for every assignment, the percentage of tasks assigned their first choice (on basis of expected execution time) is likely to be higher in Min-min than with the other batch mode heuristics described in this subsection. The expectation is that a smaller makespan can be obtained if a larger number of tasks is assigned to the machines that not only complete them earliest but also execute them fastest.

(1) **for** all tasks t_i in meta-task M_v (in an arbitrary order)
(2) **for** all machines m_j (in a fixed arbitrary order)
(3) $c_{ij} = e_{ij} + r_j$
(4) **do** until all tasks in M_v are mapped
(5) for each task in M_v find its earliest completion time and the machine that obtains it
(6) find the task t_k with the minimum earliest completion time
(7) assign task t_k to the machine m_l that gives the
(8) earliest completion time
(9) delete task t_k from M_v
(10) update r_l
(11) update c_{il} for all t_i in M_v
(12) **enddo**

Fig. 2. The Min-min heuristic.

The Max-min heuristic is similar to the Min-min heuristic given in Fig. 2. It was one of the heuristics implemented in SmartNet [13]. Once the machine that provides the earliest completion time is found for every task, the task t_k that has the maximum earliest completion time is determined and then assigned to the corresponding machine. The matrix c and vector r are updated and the above process is repeated with tasks that have not yet been assigned a machine.

The Max-min is likely to do better than the Min-min heuristic in the cases where there are many more shorter tasks than the long tasks. For example, if there is only one long task, Max-min will execute many short tasks concurrently with the long task. The resulting makespan might just be determined by the execution time of the long task in these cases. Min-min, however, first finishes the shorter tasks (which may be more or less evenly distributed over the machines) and then executes the long task, increasing the makespan compared to the Max-min.

The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would 'suffer' most in terms of expected completion time if that particular machine is not assigned to it [22]. Fig. 3 shows the Sufferage heuristic. Let the sufferage value of a task t_i be the difference between its second earliest completion time (on some machine m_y) and its earliest completion time (on some machine m_x). That is, using m_x will result in the best completion time for t_i, and using m_y the second best. The Sufferage heuristic attempts to assign each task based on the MCT. When there is a conflict for a machine, the task with the higher sufferage value is assigned to that machine.

6.5. Sample comparisons for dynamic mapping heuristics

For many heuristics, there are control parameter values and/or control function specifications

that can be selected for a given implementation. For the studies here, such values and specifications were selected based on experimentation and/or information in the literature. The above is also true for the static mapping heuristics presented in Section 7. In Figs. 4 and 5, vertical lines at the tops

```
(1)  for all tasks t_k in meta-task M_v (in an arbitrary order)
(2)      for all machines m_j (in a fixed arbitrary order)
(3)          c_kj = e_kj + r_j
(4)  do until all tasks in M_v are mapped
(5)      mark all machines as unassigned
(6)      for each task t_k in M_v (in an arbitrary order)
(7)          find machine m_j that gives the earliest
                 completion time
(8)          sufferage value = second earliest completion
                 time − earliest completion time
(9)          if machine m_j is unassigned
(10)             assign t_k to machine m_j, delete t_k
                 from M_v, mark m_j assigned
(11)         else
(12)             if sufferage value of task t_i already
                 assigned to m_j is less than the
                 sufferage value of task t_k
(13)                 unassign t_i, add t_i back to M_v,
                     assign t_k to machine m_j,
                     delete t_k from M_v
(14)     endfor
(15)     update the vector r for the tasks that
             were assigned to the machines
(16)     update the c matrix
(17) enddo
```

Fig. 3. The Sufferage heuristic.

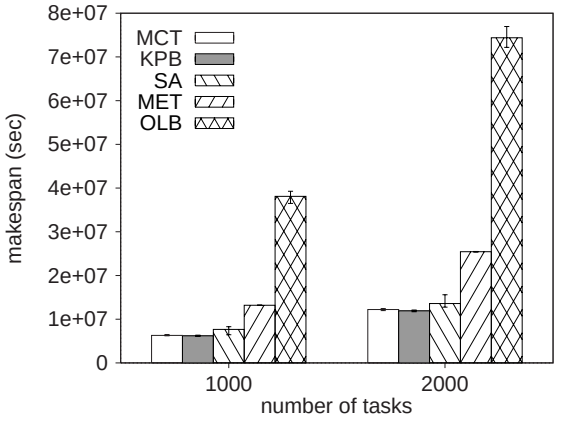


Fig. 4. Makespan for the on-line heuristics for inconsistent HiHi heterogenity, using 20 machines, based on 50 trials.

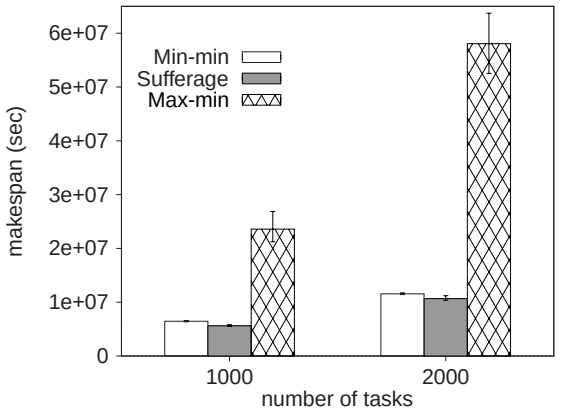


Fig. 5. Makespan of the batch heuristics for the regular time interval strategy and inconsistent HiHi heterogenity, using 20 machines, based on 50 trials.

of bars show minimum and maximum values for the 50 trials, while the bars show the averages.

In Fig. 4, the on-line mode heuristics are compared based on makespan for inconsistent HiHi heterogeneity. The KPB provides the minimum makespan, closely followed by the MCT. Fig. 5 shows the makespan for batch heuristics under similar conditions. The Sufferage heuristic gives the smallest makespan, followed by the Min-min. When the task arrival rate is relatively higher, the batch method outperformed the on-line method in these studies. The reader is referred to [22] for a detailed description of the experiments, further analysis, and more results.

7. Static heuristics for mapping meta-tasks in HC systems

7.1. Overview

This section describes and compares 11 static heuristics that can be used in an RMS like MSHN for mapping meta-tasks to machines. In a general HC system, static mapping schemes are likely to make better mapping decisions because more time can be devoted for the computation of schedules off-line than on-line. However, static schemes require that the set of tasks to be mapped be known a priori, and that the estimates of expected

execution times of all tasks on all machines be known with reasonable accuracy. A meta-task, in the context of static heuristics, is the set of all independent tasks that are being considered for mapping. Like the dynamic heuristics in Section 6, these static mapping heuristics are non-preemptive, assume that the tasks have no deadlines or priorities associated with them, and assume a dedicated HC system.

```
(1) initial population generation;
(2) evaluation;
(3) while (stopping criteria not met)
(4)     selection;
(5)     crossover;
(6)     mutation;
(7)     evaluation;
(8) endwhile
```

Fig. 6. General procedure for a genetic algorithm.

7.2. Description of static heuristics

This subsection consists of brief definitions of the eleven static meta-task mapping heuristics that are studied and fully described in [7]. The basic terms and the performance measure defined for the dynamic heuristics in Sections 6.1 and 6.2 hold for static heuristics as well, except for the terms that characterize the dynamic nature of the dynamic heuristics, e.g., fixed count strategy.

The descriptions below assume that the machine ready times are updated after each task is mapped. For cases when tasks can be considered in an arbitrary order, the order in which the tasks appeared in the ETC matrix was used.

The static opportunistic load balancing (OLB) heuristic is similar to its dynamic counterpart except that it assigns tasks in an arbitrary order, instead of order of arrival. The user directed assignment (UDA) heuristic [1] works in the same way as the MET heuristic except that it maps tasks in an arbitrary order instead of order of arrival. The fast greedy heuristic [1] is the same as the MCT, except that it maps tasks in an arbitrary order instead of their order of arrival. The static Min-min heuristic works in the same way as the dynamic Min-min, except a meta-task contains all the tasks in the system. The static Max-min heuristic works in the same way as the dynamic Max-min, except a meta-task has all the tasks in the system. The greedy heuristic performs both the static Min-min and static Max-min heuristics, and uses the better solution [1,13].

The genetic algorithm (GA) is a popular technique used for searching large solution spaces. The version of the heuristic used for this study was adapted from [29] for this particular HC environment. Fig. 6 shows the steps in a general GA [28].

The GA implemented here operates on a population of 200 chromosomes (possible mappings) for a given meta-task. Each chromosome is a $|K|$ vector, where position i $(0 \leqslant i < t)$ is the machine to which the task t_i has been mapped. The initial population is generated using two methods: (a) 200 chromosomes randomly generated from a uniform distribution, or (b) one chromosome that is the Min-min solution and 199 random chromosomes. The latter method employs the seeding of the population with a Min-min chromosome. In this implementation, the GA executes eight times (four times with initial populations from each method), and the best of the eight mappings is used as the final solution. The makespan serves as the fitness value for evaluation of the evolution.

Simulated annealing (SA) is an iterative technique that considers only one possible solution (mapping) for each meta-task at a time. This solution uses the same representation for a solution as the chromosome for the GA. SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space (e.g., [25]). This probability is based on a system temperature that decreases for each iteration. As the system temperature 'cools', it is more difficult for currently poorer solutions to be accepted.

The genetic simulated annealing (GSA) heuristic is a combination of the GA and SA techniques [26]. In general, GSA follows procedures similar to the GA outlined above. However, for the selection process, GSA uses the SA cooling schedule and system temperature, and a simplified SA decision process for accepting or rejecting new chromosomes.

The Tabu search keeps track of the regions of the solution space which have already been

searched so as not to repeat a search near these 'Tabu' areas [17]. A solution (mapping) uses the same representation as a chromosome in the GA approach. Heuristic searches are conducted within a region, and the best solution for that region is stored. Then, a new region, not on the tabu list, is searched. When a stopping criterion is reached, the best solution among regions is selected.

The final heuristic in the comparison study is known as the A* heuristic. A* is a tree-based search that has been applied to many other task allocation problems (e.g., [8,25]). The technique used here is similar to the one described in [8]. As the tree grows, intermediate nodes represent partial solutions (a subset of tasks are assigned to machines), and leaf nodes represent final solutions (all tasks are assigned to machines). The partial solution of a child node has one more task t_a mapped than the parent node. Each parent node can be replaced by its m children, one for each possible mapping of t_a. The number of nodes allowed in the tree is bounded to limit mapper execution time. Less promising nodes are deleted, and the more promising nodes are expanded. The process continues until a leaf node (complete mapping) is reached.

7.3. Sample comparisons for static mapping heuristics

Figs. 7 and 8 show comparisons of the 11 static heuristics using makespan as the criterion in two different heterogeneity environments. Vertical lines at the top of bars show minimum and maximum values for the 100 trials, while the bars show the averages. It can be seen that, for the parameters used in this study, GA gives the smallest makespan for both inconsistent HiHi and inconsistent HiLo heterogeneities. The reader is referred to [7] for more results, details, and discussions.

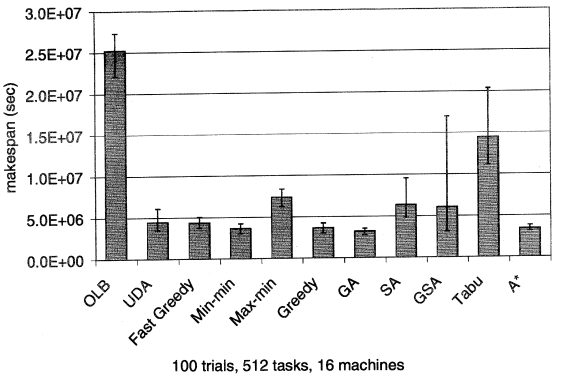


Fig. 7. Inconsistent, high task, high machine heterogenity.

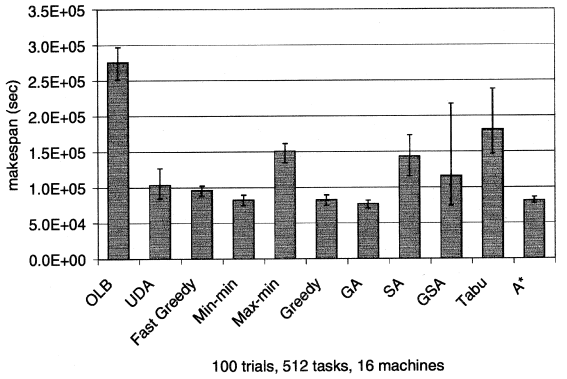


Fig. 8. Inconsistent, high task, low machine heterogenity.

8. Conclusions

Heterogeneous computing is a relatively new research area for the computer field. Interest in such systems continues to grow, both in the research community and in the user community.

Some of the different types of HC systems that have been built were discussed here, including mixed-mode, multi-mode, and mixed-machine. Mixed-machine HC was then focussed upon. A way to describe different kinds of heterogeneous environments based on characteristics of the ETC matrix was presented. The structure of a taxonomy for heuristics for mapping tasks onto mixed-machine HC systems was reviewed. As an example of the design of a resource management system for such HC environments, the high-level functional architecture of MSHN was provided. Finally, a sampling of heuristic techniques for dynamic and static mapping of independent tasks onto machines in an HC suite was given. The definition of each of these heuristics was summarized, and some example comparison results were shown. For all of these topics, references were cited where much

greater details can be found. The reader is encouraged to pursue these references to learn more.

Acknowledgements

The authors thank Tracy Braun and Surjamukhi Chatterjea for their valuable comments.

References

[1] R. Armstrong, D. Hensgen, T. Kidd, The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions, in: Seventh IEEE Heterogeneous Computing Workshop (HCW '98), March 1998, pp. 79–87.

[2] R. Armstrong, Investigation of effect of different run-time distributions on Smartnet performance, thesis, Department of Computer Science, Naval Postgraduate School, 1997 (D. Hensgen, Advisor).

[3] P.B. Bhat, V.K. Prasanna, C. S. Raghavendra, Adaptive communication algorithms for distributed heterogeneous systems, in: IEEE International Symposium on High Performance Distributed Comput., July 1998, pp. 310–321.

[4] P.B. Bhat, V.K. Prasanna, C.S. Raghavendra, Block-cyclic redistribution over heterogeneous networks, in: International Conference on Parallel and Distributed Computing Systems, Sep. 1998, pp. 242–249.

[5] P.B. Bhat, V.K. Prasanna, C.S. Raghavendra, Efficient collective communication in distributed heterogeneous systems, in: IEEE International Conference on Distributed Computing Systems, June 1999, pp. 15–24.

[6] T.D. Braun, H.J. Siegel, N. Beck, L.Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems, in: 1998 IEEE Symposium on Reliable Distributed Systems, October 1998, pp. 330–335.

[7] T.D. Braun, H.J. Siegel, N. Beck, L.Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, R.F. Freund, D. Hensgen, A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems, in: Eighth IEEE Heterogeneous Computing Workshop (HCW '99), April 1999, pp. 15–29.

[8] K. Chow, B. Liu, On mapping signal processing algorithms to a heterogeneous multiprocessor system, in: 1991 International Conference on Acoustics, Speech, and Signal Processing – ICASSP 91, vol. 3, 1991, pp. 1585–1588.

[9] M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996.

[10] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Software Engrg. SE-15 (11) (1989) 1427–1436.

[11] S.A. Fineberg, T.L. Casavant, H.J. Siegel, Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting, J. Parallel and Distributed Comput. 11 (3) (1991) 239–251.

[12] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Fransisco, CA, 1999.

[13] R.F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.D. Lima, F. Mirabile, L. Moore, B. Rust, H.J. Siegel, Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet, in: Seventh IEEE Heterogeneous Computing Workshop (HCW '98), March 1998, pp. 184–199.

[14] R.F. Freund, T. Kidd, D. Hensgen, L. Moore, SmartNet: A scheduling framework for metacomputing, in: Second International Symposium on Parallel Architectures, Algorithms, and Networks (ISPAN '96), June 1996, pp. 514–521.

[15] R.F. Freund, Optimal selection theory for superconcurrency, in: Supercomputing '89, November 1989, pp. 699–703.

[16] A. Ghafoor, J. Yang, Distributed heterogeneous supercomputing management system, IEEE Comput. 26 (6) (1993) 78–86.

[17] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, MA, 1997.

[18] D.A. Hensgen, T. Kidd, D. St. John, M.C. Schnaidt, H.J. Siegel, T.D. Braun, M. Maheswaran, S. Ali, J.-K. Kim, C. Irvine, T. Levin, R.F. Freund, M. Kussow, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, A. Alhusaini, An overview of MSHN: the Management System for Heterogeneous Networks, in: Eighth IEEE Heterogeneous Computing Workshop (HCW '99), April 1999, pp. 184–198.

[19] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, J. ACM 24 (2) (1977) 280–289.

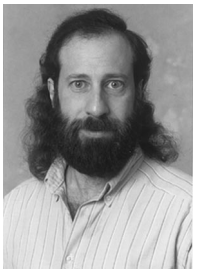
[20] A. Khokhar, V.K. Prasanna, M. Shaaban, C.L. Wang, Heterogeneous computing: Challenges and opportunities, IEEE Comput. 26 (6) (1993) 18–27.

[21] J.-K. Kim, D.A. Hensgen, T. Kidd, H.J. Siegel, D. St. John, C. Irvine, T. Levin, N.W. Porter, V.K. Prasanna, R.F. Freund, A multi-dimensional QoS performance measure for distributed heterogeneous networks, Technical Report, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, in preparation.

[22] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: Eighth IEEE Heterogeneous Computing Workshop (HCW '99), April 1999, pp. 30–44.

[23] M. Maheswaran, T.D. Braun, H.J. Siegel, Heterogeneous distributed computing, in: J.G. Webster (Ed.), Encyclopedia of Electrical and Electronics Engineering, Wiley, New York, NY, 8 (1999) 679–690.

[24] R. van Renesse, A.S. Tanenbaum, Distributed operating systems, ACM Comput. Surveys 17 (4) (1985) 419–470.
[25] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[26] P. Shroff, D.W. Watson, N.S. Flann, R.F. Freund, Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments, in: Fifth IEEE Heterogeneous Computing Workshop (HCW '96), April 1996, pp. 98–117.
[27] H.J. Siegel, M. Maheswaran, D.W. Watson, J.K. Antonio, M.J. Atallah, Mixed-mode system heterogeneous computing, in: M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996, pp. 19–65.
[28] M. Srinivas, L.M. Patnaik, Genetic algorithms: a survey, IEEE Comput. 27 (6) (1994) 17–26.
[29] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel and Distributed Comput. 47 (1) (1997) 1–15.
[30] C.C. Weems, S. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, The image understanding architecture, Internat. J. Comput. Vision 2 (3) (1989) 251–282.



Howard Jay Siegel is a Professor in the School of Electrical and Computer Engineering at Purdue University. He is a Fellow of the IEEE and a Fellow of the ACM. He received BS degrees in both electrical engineering and management from MIT, and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. Prof. Siegel has coauthored over 250 technical papers, has coedited seven volumes, and wrote the book *Interconnection Networks for Large-Scale Parallel Processing*. He was a Coeditor-in Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three conferences, General Chair/Co-Chair of four conferences, and Chair/Co-Chair of four workshops. He is an international keynote speaker and tutorial lecturer, and a consultant for government and industry.

Shoukat Ali is pursuing a Ph.D. degree from the School of Electrical and Computer Engineering at Purdue University, where he is currently a Research Assistant. His main research topic is dynamic mapping of metatasks in heterogeneous computing systems. He has held teaching positions at Aitchison College and Keynesian Institute of Management and Sciences, both in Lahore, Pakistan. He was also a Teaching Assistant at Purdue. Shoukat received his BS degree in electrical and electronic engineering from the University of Engineering and Technology, Lahore, Pakistan in 1996. He received his M.S.E.E. degree from the School of Electrical and Computer Engineering of Purdue University in 1999. His research interests include computer architecture, parallel computing, and heterogeneous computing.